Back-end Automated Code Review (Sprint 2)

Due to technical issues and time constraints, we did not use automated code reviews in this sprint. Instead, as mentioned by Ed, we opted for a manual approach using ChatGPT to review our code. Moving forward, we will continue troubleshooting and aim to use automated ChatGPT code reviews in Sprint 3.

Introduction

Purpose

The purpose of this document is to outline our code review process which plays a crucial role in maintaining high code quality and identifying potential issues early. Code reviews help ensure that our software is robust, scalable, and secure.

Scope

This documentation covers the automated code review processes for the SciGallery project, focusing on backend services and critical frontend components.

Code Review Process

Participants

- **YUN DAI -** Backend developer (Code Submitter)
- **Qingyuan Yang** - Backend developer
- **AI Reviewer** - ChatGPT 4.0

Frequency

Code reviews are conducted bi-weekly before each sprint ends, or after significant commits to our Git repository, such as new feature additions or critical bug fixes.

Selection Criteria

Criteria for Code Selection

Code is selected for review based on several factors including:

- Complexity of the code.
- The criticality of the application area affected.
- Recentness of changes.
- Modules with a history of frequent issues.

Tools and Techniques Used

ChatGPT 4.0

Review Outcomes

Number of Reviews Conducted

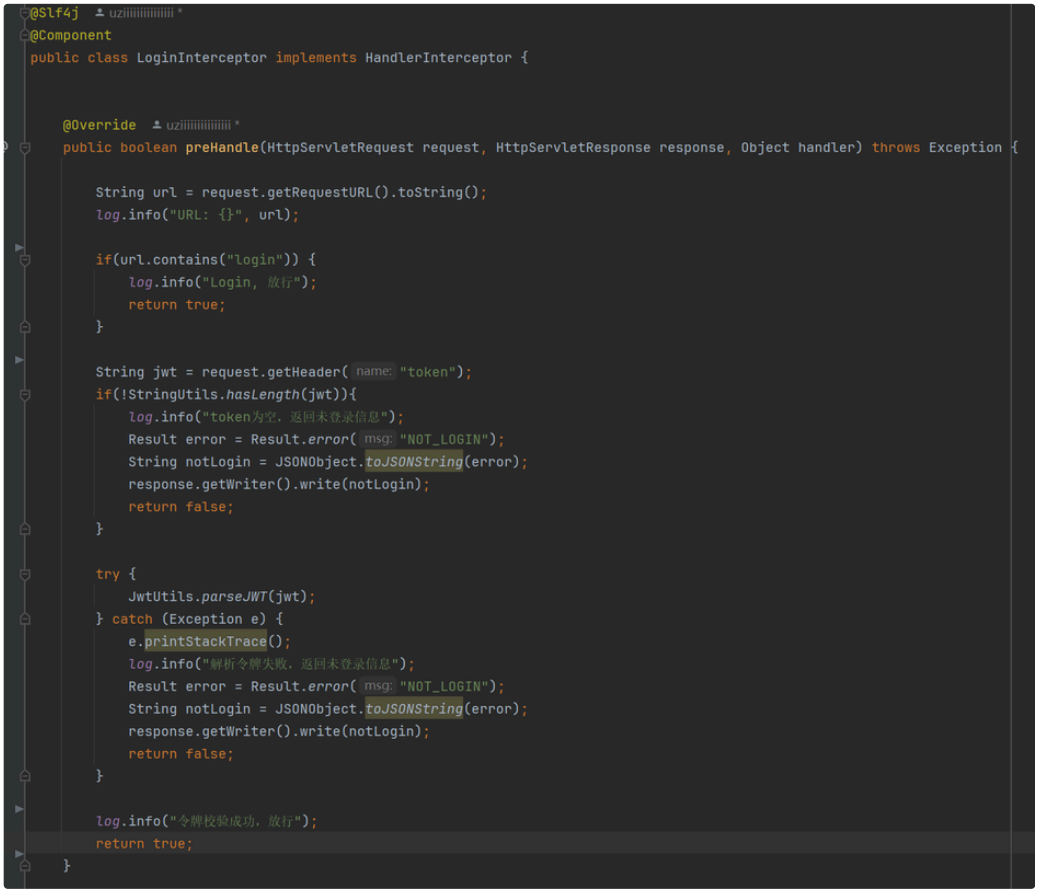
In the last quarter, we conducted 12 code reviews.

Issues Identified

A total of 12 issues were identified.

Example of one of the code review:

Code:



```java
@Slf4j
@Component
public class LoginInterceptor implements HandlerInterceptor {


    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

        String url = request.getRequestURL().toString();
        log.info("URL: {}", url);

        if(url.contains("login")) {
            log.info("Login, 放行");
            return true;
        }

        String jwt = request.getHeader( name: "token");
        if(!StringUtils.hasLength(jwt)){
            log.info("token为空，返回未登录信息");
            Result error = Result.error( msg: "NOT_LOGIN");
            String notLogin = JSONObject.toJSONString(error);
            response.getWriter().write(notLogin);
            return false;
        }

        try {
            JwtUtils.parseJWT(jwt);
        } catch (Exception e) {
            e.printStackTrace();
            log.info("解析令牌失败，返回未登录信息");
            Result error = Result.error( msg: "NOT_LOGIN");
            String notLogin = JSONObject.toJSONString(error);
            response.getWriter().write(notLogin);
            return false;
        }

        log.info("令牌校验成功，放行");
        return true;
    }
```

ChatGPT Review:

ChatGPT

Evaluation of the Provided Code

1. **Documentation Defects**
   - **a. Naming:**
     - The naming conventions are generally clear, such as `` `jwt` ``, `` `error` ``, and utility method names like `` `parseJWT` ``. However, the log messages could be more descriptive regarding what is specifically happening, such as what it means when a token is "put through" or when a login is "allowed".
   - **b. Comment:**
     - The code lacks comments explaining the purpose and functionality of the methods and important blocks within the `` `preHandle` `` method. Adding explanatory comments, especially about the JWT validation process, would enhance maintainability and readability.
2. **Visual Representation Defects**
   - **a. Bracket Usage:**
     - Brackets are used correctly throughout the code.
   - **b. Indentation:**
     - The indentation is consistent, making the code structure clear and readable.
   - **c. Long Line:**
     - The lines where JSON objects are converted to strings (`` `JSONObject.toJSONString(error)` ``) could be broken down for better readability.
3. **Structure Defects**
   - **a. Dead Code:**
     - There does not appear to be dead code in the provided snippet.
   - **b. Duplication:**
     - The creation of the `` `Result` `` object and its conversion to JSON string is duplicated in the JWT parsing catch block and the check for JWT presence. This could be refactored into a separate method.
4. **New Functionality**
   - **a. Use Standard Method:**
     - The exception handling in JWT parsing uses generic exception catching which might not be the most effective way to handle specific JWT parsing errors. More specific exception handling could improve error diagnostics.
5. **Resource Defects**
   - **a. Variable Initialization:**
     - Variables are correctly initialized in their scope.
   - **b. Memory Management:**

5. **Resource Defects**
   - **a. Variable Initialization:**
     - Variables are correctly initialized within their scope.
   - **b. Memory Management:**
     - The code does not explicitly manage memory, relying on Java's garbage collector. There is no evident misuse of resources.
6. **Check Defects**
   - **a. Check User Input:**
     - The code correctly checks for the JWT in the header but does not perform any deeper validation of the JWT contents (like checking claims or issuer). Expanding on these checks would improve security.
7. **Interface Defects**
   - **a. Parameter:**
     - There are no interface defects evident from the code provided, as the method signatures align with the Spring framework's expectations.
8. **Logic Defects**
   - **a. Compute:**
     - The logic for handling the request path and JWT seems correct.
   - **b. Performance:**
     - The code performs minimal operations in terms of computational overhead, though optimization could be made by reducing duplication and handling specific exceptions more effectively.

Summary of Recommendations:

- **Add Comments:** Enhance the code with comments explaining each major step, especially around JWT validation.
- **Refactor Duplication:** Extract the repeated logic for error handling into a separate method to reduce duplication and improve code manageability.
- **Enhance JWT Validation:** Implement more robust JWT validation, checking for claims, issuer, and possibly expiration, depending on security requirements.
- **Improve Error Handling:** Use more specific exceptions for JWT parsing to enhance error diagnostics and potentially handle different error conditions differently.
- **Optimize JSON Handling:** Consider refactoring the JSON serialization into a method if used frequently throughout the application, to maintain consistency and reduce potential errors.

Feedback and Actions

Review of AI Feedback

Feedback from ChatGPT was reviewed by the development team within one day of the automated review completion. Priority was given to addressing critical vulnerabilities first.

Actions Taken After Review

Changes Implemented

- **Add Comments:** Enhance the code with comments explaining each major step.
- **Refactor Duplication:** Extract the repeated logic for error handling into a separate method to reduce duplication and improve code manageability.
- **Code smells:** Refactoring was performed to improve code readability and maintainability.

Unchanged Code and Reasons

- During our recent code review, it was noted that there are updated solutions available for some of the technologies and methodologies we are currently using. However, after considering the time required to adopt these new solutions, we have decided to maintain our existing technology stack.

Evaluation and Continuous Improvement

Effectiveness of Reviews

The reviews significantly reduced the number of runtime errors and improved the security posture of the application.

Lessons Learned and Improvements

We learned the importance of immediate action on critical issues and plan to integrate more detailed checklists for reviewers to cover edge cases better in future reviews.

Conclusion

The automated code review process has been instrumental in maintaining the quality and security of our software. Ongoing improvements and adaptations will continue to refine our review processes.